Exploring Improvements to the SQuAD 2.0 BiDAF Model

Stanford CS224N Default Project

Bihan Jiang
Department of Computer Science
Stanford University
bihan@stanford.edu

Lauren Yang
Department of Computer Science
Stanford University
lauren11@stanford.edu

Aakanksha Saxena
Department of Computer Science
Stanford University
asaxena1@stanford.edu

Abstract

We have explored different deep learning based approaches to the question answering problem on SQuAD 2.0 using an improved version of the BiDAF model. Our baseline was provided by the default project starter code, and is a modified BiDAF that has only word embeddings and performs on SQuAD 2.0. We explored three areas of improvements: character embeddings, conditioning the end prediction on the start prediction, and adding a self-attention layer. We found that the biggest improvement was from the **Condition End Prediction on Start Prediction and Self-Attention** with an F1 and EM score of **65.285 and 61.758** on the test set respectively.

1 Introduction and Related Work

Question answering (Q&A) using unstructured texts is very useful as reading comprehension is an important test for evaluating how well computer systems understand human language and because many other NLP tasks can often be reduced to reading comprehension problems. This project focuses on the Stanford Question Answering Dataset 2.0 (SQuAD 2.0), which has about 150k (context, question, answer) data points from Wikipedia articles [1]. The goal of the Q&A task is to select the answer span from the context in order to answer the question at hand.

In recent years, Q&A with neural networks has been evaluated thoroughly. The baseline model that we are provided is an implementation of the Bidirectional Attention Flow (BiDAF) architecture, which combines representations of the context and question and has state-of-the-art results [1]. The goal of this project is to improve the performance of the baseline model for the SQuAD 2.0 task, using character embeddings, self-attention, and conditioning end prediction on start prediction.

Character Embeddings: The original BiDAF paper [1] includes character embeddings, but the baseline implementation that we were provided does not. The embedding layer currently does an embedding lookup, translating question and context indices into word embeddings. Character embeddings allow for conditioning on the internal structure of words and better handling of out-of-vocabulary words [2]. We re-implement this to allow the baseline implementation to match the original paper.

Self Attention: Implementing self attention will enable the model to capture understanding between context-to-context, allowing the model to extract information from the whole context given the current information from the context and question. Our implementation of self-attention is inspired by Natural Language Computing's R-net paper, and will be guided by the approach they have outlined

[3]. Specifically, we will introduce a 2D matrix to represent the embeddings, where each row attends to a different part of the sentence. The paper cites a performance of 71.1 EM score and a 79.5 F1 score for a single model, which is an improvement over the original BiDAF model. We hope to emulate their approach and achieve similar levels of performance.

Condition end prediction on start prediction: We will condition the end prediction on the start prediction by implementing both the sequence model of the Answer Pointer Layer as described in Wang et al (2017)'s paper and compare the performance [4]. This re-implementation will be completed with guidance from Wang et al (2017)'s paper [4].

2 Approach

2.1 Baseline

The baseline we are using is an implementation based on Seo et. al (2016)'s paper [2]. The baseline is a modified Bidirectional Attention Flow (BiDAF), where the model has only word embeddings and no character embeddings. The BiDAF has the following layers: word embedding layers, a contextual embedding layer that applies a bi-directional LSTM to the context and question, an attention layer that connects question to context along with context to question, a modeling layer, and an output layer. Please refer to this default project's project description for more detail regarding the provided baseline.

2.2 Character Embeddings

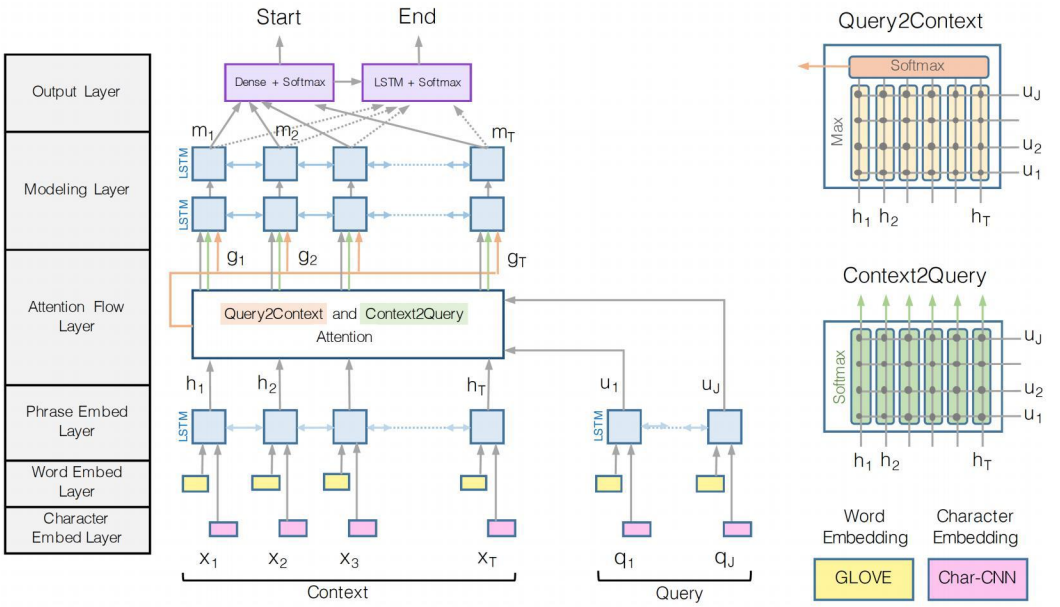

Figure 1: This shows the original BiDAF, including the character embeddings layer.

As **Figure 1** shows, to improve on the baseline, we implemented character embeddings with the provided modified BiDAF model. With this extension, the model has both word and character embeddings, which enables better performance as seen in **Table 1**. This is because character embeddings allow the model to learn from internal morphological structures of words and better handle out of vocabulary words. Additionally, words that are misspelled, and words that do not have a word embedding can be better interpreted with character embeddings. This extension is inspired by the original BiDAF implementation, which has both types of embeddings [2].

In order to implement this, the characters of individual words were mapped to a lower dimensional space and assigned a unique integer to represent every unique character. After this, each character token is mapped to an embedding layer, which produces a mapping such that for a word of length l and character tokens $c_1, c_2, c_3 ... c_l$, an embedding is produced, $e_1, e_2, e_3 ... e_l$. The ith embedding of the character, e_i, has a vector size of d_c, and thus, a word of length l_w would have

an output matrix size of l_w x d_c, and a sentence of length l_s would have an output matrix of size l_sx d_c.

These character level embeddings are passed through a 1D convolutional neural network (CNN), and a hidden representation of the characters is produced. If the hidden layer is $h_1, h_2, ...h_l$, then h_i represents the hidden representation of the ith character, and h_i is calculated by convoluting the character embeddings $e_{i-\frac{k}{2}}, e_{i+1-\frac{k}{2}}, ...e_i, ...e_{i-1+\frac{k}{2}}, e_{i+\frac{k}{2}}$, where k is the filter size. The result of this operation is a vector with length v, and the resulting output matrix from this layer is of size l_s x l_w x v. Each word's output matrix is max pooled, and the character embeddings are concatenated word embeddings to ultimately create a l_s x $(d_w + f)$ matrix, where d_w is the size of the word embeddings.

2.3 Self-Attention

In addition to the baseline and character embeddings, we have implemented a modification that utilizes self-attention. We have implemented the extension on top of the provided BiDAF model. In the baseline implementation, the sentence embedding is represented by vectors, and includes the bi-directional attention flow, which captures the context-query and query-context attentions. One issue with this is that question-aware context has limited knowledge of previous context.

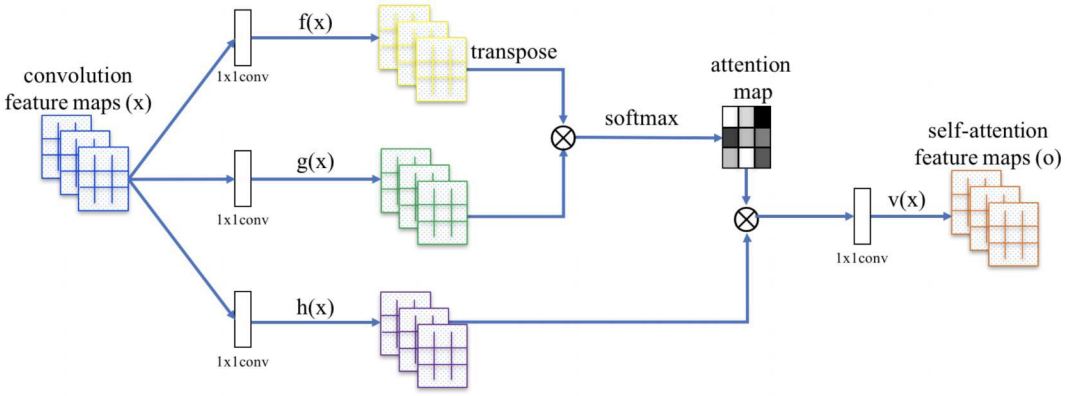

Figure 2: Self-attention layer

In our implementation of self-attention, we matched the question-aware passage representation with itself [3]. We added another attention layer that computes the attention scores between context-to-context by taking the attention score tensor from the baseline. We then added a non-linear activation function and dropout layer, and we used a softmax function to normalize the scores. The resulting attention scores had the same dimensions as the query-context attention, so the results could be concatenated in a similar way.

The self-attention layer extracts information from the whole context according to the current information of context and question. The is helpful for the model because it can extract referential relationships in the long contexts that require more complex reasoning. Specifically, we calculate the attention scores of each word representation M_t with all the other word representations M_j and obtain the final weighted sum representation of each word.

$$z_j^t = v^T tanh(W_m^P M_j + W_m^{\tilde{P}} M_t) \tag{1}$$

$$a_i^t = exp(z_i^t) / \sum_{j=1}^{T} exp(z_j^t) \tag{2}$$

$$P_t = \sum_{i=1}^{T} a_i^t M_i \tag{3}$$

In the equations above, v^T, W_m^P and $W_m^{\tilde{P}}$ are trainable weights, and P is the final self-attention context representation. The output P of the self-attention flows directly into the output layer.

2.4 Conditioning End Prediction on Start Prediction

The baseline model currently predicts the start location and location for answers independently in the output layer. We have modified the output layer to condition the probability distribution for the end location on the start location's probability distribution. We implemented the Answer Pointer component of the 'Match-LSTM with Answer Pointer' model [4]. We used the boundary model, where a_e is conditioned on a_s and the concatenated input H, as shown in equation 4. We start out with attention distribution $\beta_s \in \mathcal{R}^N$, which represents the probability distribution for the start location, and an attention output $a_s \in \mathcal{R}^l$. We use the results of β_s and a_s to calculate $\beta_e \in \mathcal{R}^N$, which represents p_{end} using the Answer Pointer RNN method.Using equation 5, p_{end} is used to produce the end location.

$$p(a|H) = p(a_s|H)p(a_e|a_s, H) \tag{4}$$

$$l_{start,end} = argmax_{j,i} p_{j,start} * p_{i,end} \tag{5}$$

2.5 Model Diagram

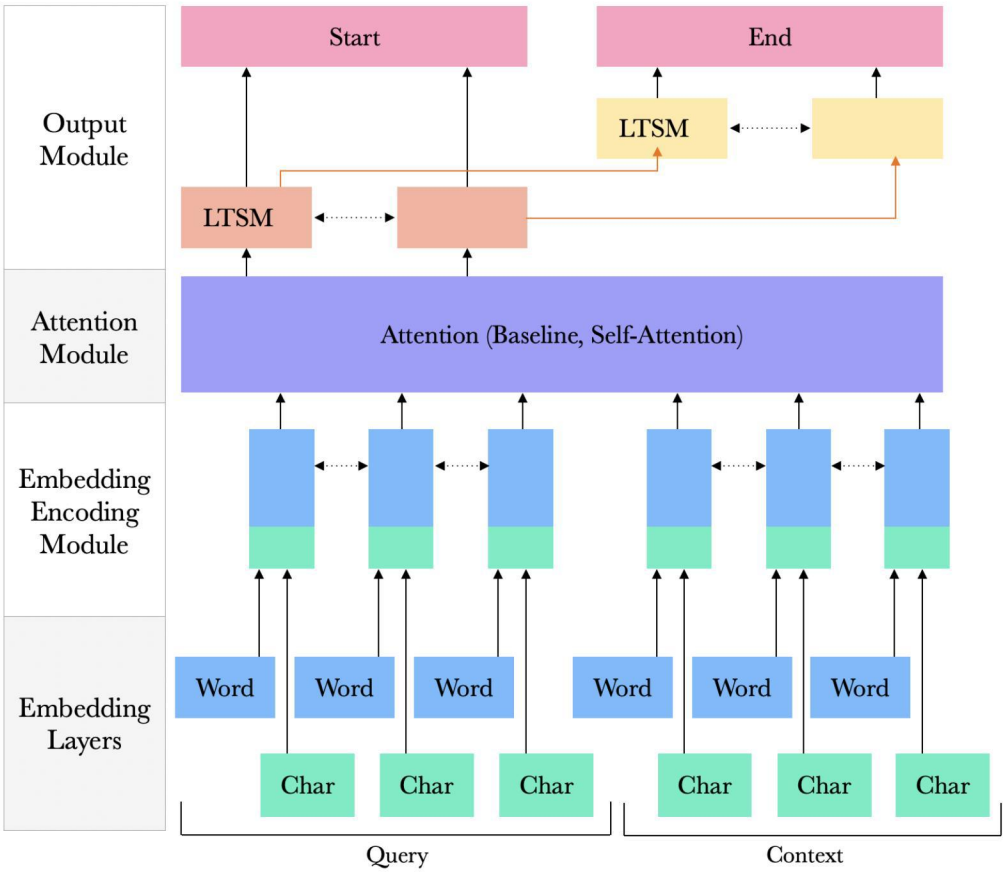

Figure 3: Our overall model architecture, with the extensions outlined.

3 Experiments

3.1 Data

We are utilizing the public train, test, and dev datasets available on the public SQuAD 2.0 dataset. The data is structured as over 150k context, question, answer combinations on more than 500 articles. Additionally, the dataset contains more than 50000 unanswerable questions. To perform well on the SQuAD 2.0 dataset, models must both answer questions correctly when possible, as well as determine when no answer is supported by the text and respond with "no answer". Context comes from Wikipedia articles, and the answer is a direct excerpt from the context or "No answer". We did not need to use significant preprocessing as the data has already been aggregated.

3.2 Evaluation method

We are using two metrics to evaluate model performance: exact match (EM) and F1. EM is measures the degree to which the output exactly matches the human answer. The F1 score is the harmonic mean of precision and recall, and is calculated as so: $F_1 = \frac{2*p*r}{(p+r)}$ where $p =$ precision and $r =$recall. We implemented each of our extensions as a separate model. We are comparing the results of these models with the default baseline model that is provided in the starter code. Essentially, we are benchmarking against each original method's performance.

3.3 Experimental details

We have set the default learning rate as 0.5 for all of our models and use weight decay. We added a model_selected flag that allows the user to select which model they would like to run. To run both self-attention, character embeddings, and conditioning end probability on start probability, model_selected should be set as "all_extensions". Each model was trained for 12-20 hours on a NC6_Promo.

One easy way to improve the performance of our model is to add character-level embeddings to the baseline models, after which an increase in F1 and EM scores were observed. Then, a self-attention layer was added after the BiDAF layer to further boost the performance. Since adding another layer to the model increased the complexity of the model, the training time increased to around 20 hours. Finally, we changed the output layer to condition end prediction on start prediction. The results are shown in **Table 1**.

3.4 Results

Table 1: Results

Method/Implementation	EM	F1
Baseline (original BiDAF)	57.70	61.00
Character Embeddings	59.96	63.24
Character Embeddings and Self-Attention	63.00	66.20
Condition End Prediction on Start Prediction and Self-Attention, dev	63.23	66.27
Condition End Prediction on Start Prediction and Self-Attention, test	61.758	65.285

We achieved an F1 score of **65.285** and an EM score of **61.758** on the provided SQuAD test set for the IID SQuAD track. Compared to our dev set results, these were a bit lower than expected. Detailed results from all our modifications are shown in the table above. We see that our best EM and F1 scores on the dev set are from conditioning the end prediction on the start prediction, with an EM score of 63.23 and an F1 score of 66.27.

In terms of what we expected with results, we expected that the scores for the character embeddings plus the self-attention layer would be higher than the scores for the baseline model as well as the baseline + character embeddings model. However, we expected the self-attention extension to significantly improve the performance of the model compared to the actual improvements we achieved. Observing the results, we see indeed that by implementing self-attention in tandem with character embeddings, the EM score improved by 5.3 and the F1 score improved by 5.2 in comparison to the baseline. In comparison to the model with just character embeddings, the EM score improved by 3.04 and the F1 score improved by 2.96. We were also surprised conditioning the end prediction on the start prediction was only sightly better than the model with character embeddings and self-attention. This is because in the original paper written by Wang et. al (2017), which proposed this method, achieved an EM score of 64.1 on the dev set and 64.7 on the test set, and a F1 score of 73.9 on the dev set and 73.7 on the test set [4]. We believe that implementing teacher forcing on this model, so that the correct start word is used to select the end word during training, would be the next step for improving results and moving towards the results of the paper [5]. Compared to our results, we achieved a score of 63.23 on the EM score and a 66.27 on the F1 score.

4 Analysis

Both the F1 and EM scores showed improvements with each additional feature we added to our model. Compared to the baseline model, the combined condition end prediction on start prediction, self-attention, and character embeddings model obtains better scores than the baseline, character-embeddings, and character embedding + self-attention models.

4.1 Error Analysis

In this section, we analyze the performance of our best model, which used conditioning the end prediction on start prediction and self-attention. Particularly, we investigate where the model had errors, how those errors came to be, and potential improvements.

4.1.1 Lack of Contextual Knowledge

Example 1

Context, excerpt: Gothic architecture is represented in the majestic churches but also at the burgher houses and fortifications... The most interesting examples of **mannerist architecture** are the Royal Castle (1596,Äì1619) and the Jesuit Church (1609,Äì1626) at Old Town.

Question: What is the Royal Castle the most interesting example of?

True Answer: mannerist architecture

Predicted Answer: N/A

Analysis: For this question, the predicted answer, "mannerist architecture", appears near a key term of the question, "the Royal Castle". However, the model is most likely incorrectly parsing the dependencies in the context, and associating "example" with "mannerist architecture", and "Royal Castle" with "Jesuit Church". The model does not have the contextual knowledge to understand that "Royal Castle" with "Jesuit Church" are examples of architectural accomplishment, and is thus unable to produce a prediction.

4.1.2 Lack of Syntactical Understanding

Example 2

Context, excerpt: Thanks to numerous musical venues, including the Teatr Wielki, the Polish National Opera, the Chamber Opera, the National Philharmonic Hall and the National Theatre, as well as the Roma and Buffo music theatres and the Congress Hall **in the Palace of Culture and Science**, Warsaw hosts many events and festivals.

Question: Where is the Congress Hall located?

True Answer: in the Palace of Culture and Science

Predicted Answer: Palace of Culture and Science, Warsaw

Analysis: Although the predicted answer is very similar to the true answer in this case, it is not exactly the true answer as it is missing the word "in". This suggests the model lacks an understanding of syntax. The model did correctly include "Palace of Culture and Science" in the the predicted context span, but was unable to understand that the answer is a prepositional phrase beginning with "in". This might be because this is a complex dependency relationship that cannot be captured by the self attention we implemented. To improve, we could implement vector representations of predicted despondency parsings in order to capture with higher fidelity such relationships.

4.1.3 Incorrect Inference

Example 3

Context, excerpt: Starting in the late 1950s, American computer scientist Paul Baran developed the concept Distributed Adaptive Message Block Switching with the goal to provide a fault-tolerant, efficient routing method for telecommunication messages as part of a research program at the RAND Corporation, funded by the US Department of Defense. This concept contrasted and contradicted the theretofore established principles of pre-allocation of network bandwidth, largely fortified by the development of telecommunications in the Bell System.

Question: What was the Bell System?

True Answer: N/A

Predicted Answer: pre-allocation of network bandwidth, largely fortified by the development of telecommunications

Analysis: Here, the model is incorrectly attending "Bell System" with the phrase directly before "Bell System" as it appears in the passage, and predicts that Bell System's definition appears right before the term in the context. However, the model does not notice the fact that in the passage, Bell Systems was never defined, and brought up to contrast the system Paul Baran invented earlier in the passage.

4.1.4 Question Type Analysis

Table 2: Analysis of Model Performance on Question Type*

Model	Score	What	Who	When	How	In	Where	Which	The
Char Embeddings	EM	0.47	0.53	0.63	0.49	0.51	0.52	0.54	0.48
Char Embeddings	F1	0.5	0.56	0.64	0.53	0.52	0.58	0.6	0.51
Conditioning	EM	0.46	0.53	0.62	0.48	0.51	0.49	0.6	0.46
Conditioning	F1	0.49	0.55	0.63	0.52	0.53	0.54	0.65	0.49
Char Embed + Attn	EM	0.46	0.52	0.62	0.48	0.51	0.52	0.52	0.44
Char Embed + Attn	F1	0.49	0.54	0.64	0.52	0.54	0.57	0.57	0.49

*Dark purple means lower values, and light purple indicates higher values.

Overall, it appears that all models seem to have similar performance patterns across different question types. All models appear to do very well with questions that begin with "when", as the range for EM and F1 is between .62 and .64. This is most likely because questions involving temporal aspects are more straightforward, involve less complicated dependency parsings, and contain less syntactical knowledge.

Models tend to perform poorly when the question begins with "what" and "the". This might be because questions of this nature require an understanding of complex dependency relationships or syntactical or contextual knowledge. Examining a sample question from the dev set that begins with "the" might explain why performance is lower. Questions like "The time required to output an answer on a deterministic Turing machine is expressed as what", and "The complexity of problems often depends on what", are structured in a less straightforward way as "when" questions, and thus may cause lower performance.

We begin to see more stratified performance with "where" and "which" questions. The character embeddings model performed the best with the "where" questions, and the model which conditioned end prediction on start prediction performed the worst with such questions. Regarding the "which" questions, the model with prediction conditioning performed the best, and the model with character embeddings and self attention performed the worst with questions of this type.

5 Conclusion

In this project, we explored a combination of methods for the question-answering task on the SQuAD 2.0 dataset. We implemented character-embeddings, self-attention, and conditioning end-prediction on start-prediction, and each new feature achieved higher performance compared to the baseline model.

The best scores were obtained from the model that conditioned end predictions on start predictions, achieving a F1 and EM score of 65.285% and 61.758% respectively for the test set, and 66.27% and 63.23% respectively on the dev set. Looking forward, if we have more computational power and longer training periods, we will fine-tune our model further and test the results with different hyperparameters. Additionally, we can also add pre-trained ELMO embeddings, which should improve the accuracy of the model even further [6]. Beyond these improvements, we also hope to explore future directions like implementing co-attention, adding additional input features, and applying transformers [7] [8]. We would like to explore the performance of different extensions combined with one another.

Overall, this project was very educational, as it allowed us to read through numerous papers that outlined breakthrough improvements to this problem, and enabled us to implement ourselves the methods described in the papers. This re-implementation aspect greatly helped solidify our understanding of the reserach approaches. As we conclude this final project, we would like to give a huge thank you to our project mentor for being supportive as well as the entire CS224N teaching staff.

References

[1] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

[2] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[3] Natural Language Computing Group. R-net: Machine reading comprehension with self-matching networks. May 2017.

[4] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *CoRR*, abs/1608.07905, 2016.

[5] Alex Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks, 2016.

[6] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.

[7] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604, 2016.

[8] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *CoRR*, abs/1804.09541, 2018.